

Федеральное государственное бюджетное образовательное учреждение высшего образования



**«Московский государственный технический
университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ ФУНДАМЕНТАЛЬНЫЕ НАУКИ
КАФЕДРА МАТЕМАТИКА И КОМПЬЮТЕРНЫЕ НАУКИ

Отчет

по лабораторному заданию № 18

Вариант 18

Дисциплина: Информатика

Название лабораторного задания: Программирование с использованием
бинарных деревьев.

Студент гр. ФН11-22Б

Хе 31.06.21
(Подпись, дата)

М.Х. Хаписов
(И.О. Фамилия)

Преподаватель Доцент кафедры ИУ-6

Т.Н. Ничушкина
(Подпись, дата)

Т.Н. Ничушкина
(И.О. Фамилия)

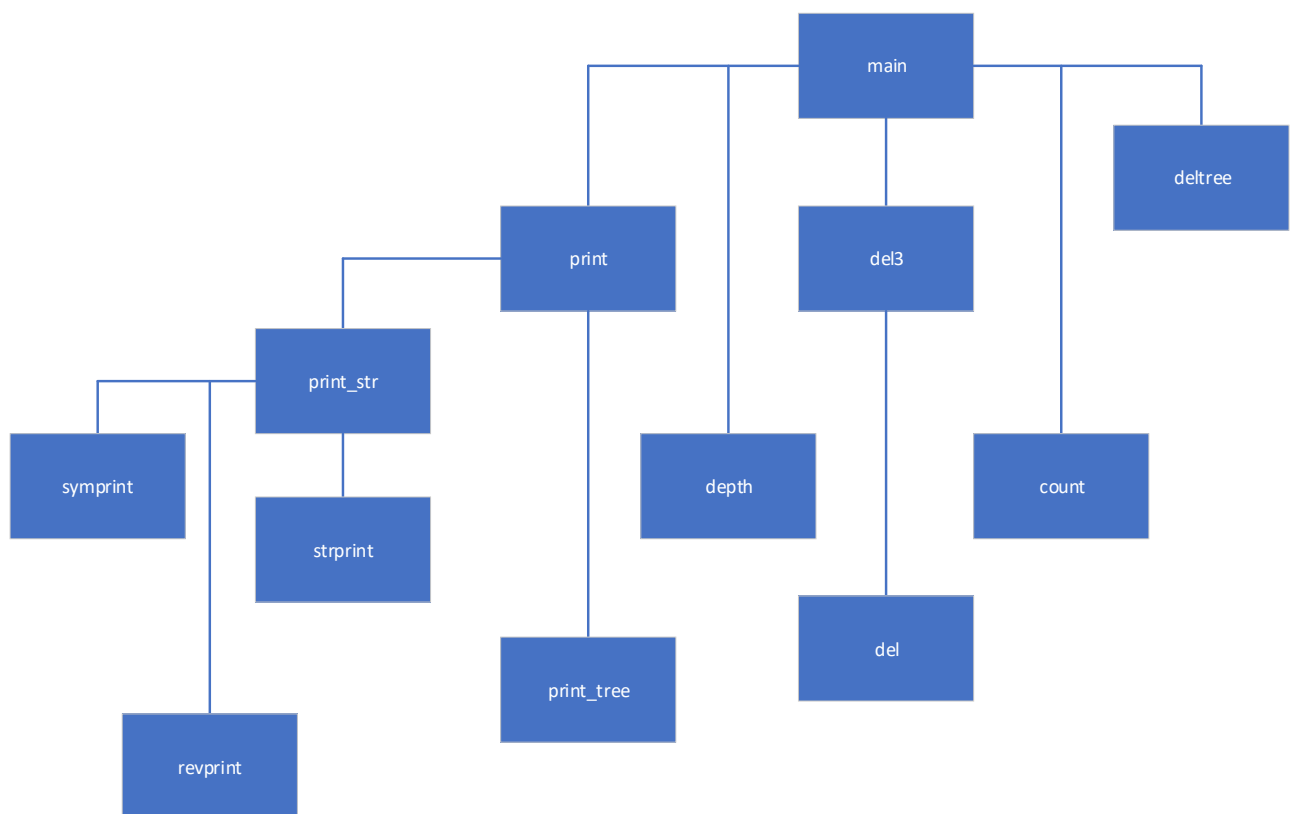
1.06.2021

Москва, 2021

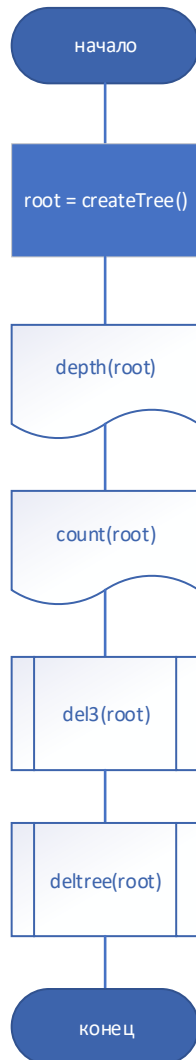
Цель работы: изучение принципов и приёмов работы с динамическими структурами данных на примере бинарных деревьев поиска

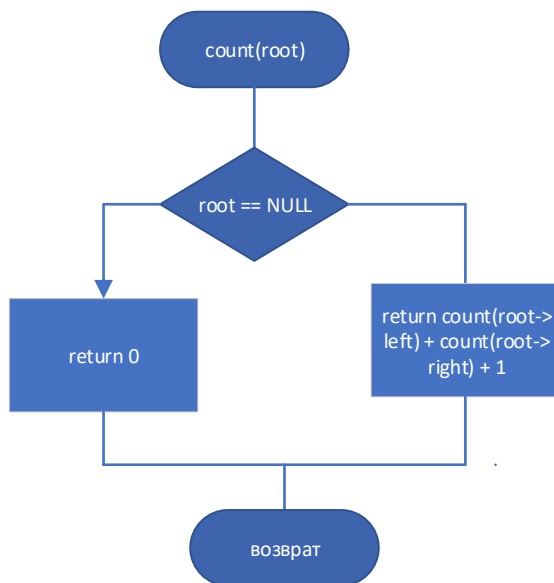
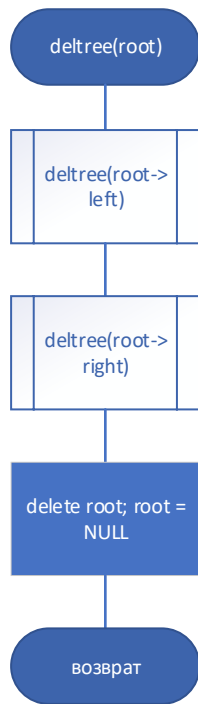
Задание: Построить двоичное дерево поиска из вводимой последовательности чисел, располагая числа с учетом их значения. Вывести дерево на экран. Определить все числа, кратные 3 и их количество. Удалить из дерева эти числа. Вывести оставшиеся элементы дерева. Реализовать подпрограмму добавления нового числа в список пожеланию пользователя. Вывести на экран всю необходимую информацию с комментариями.

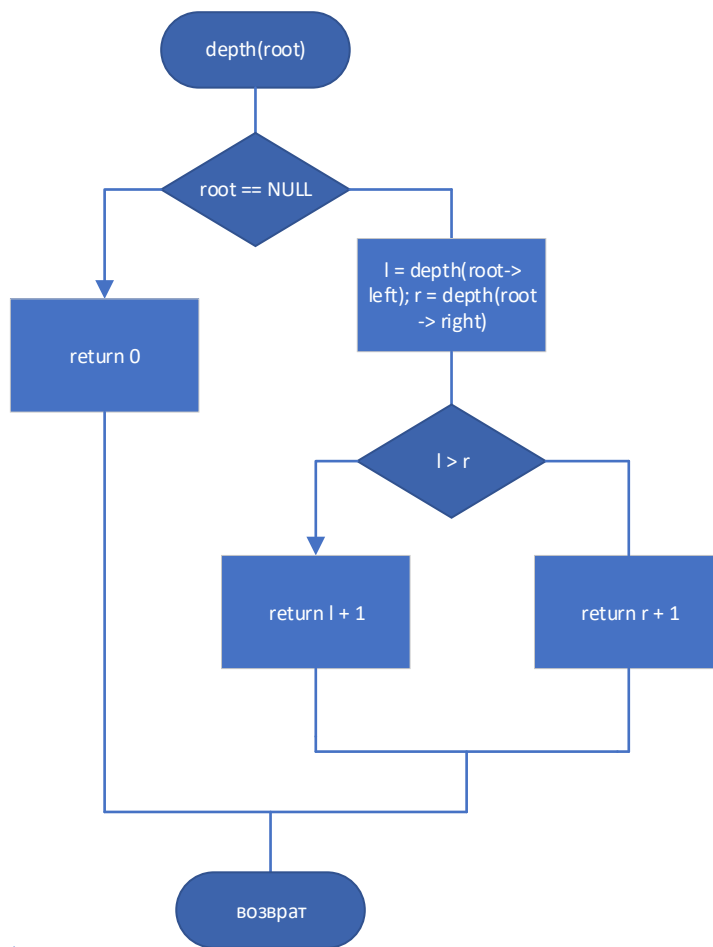
Структурная схема

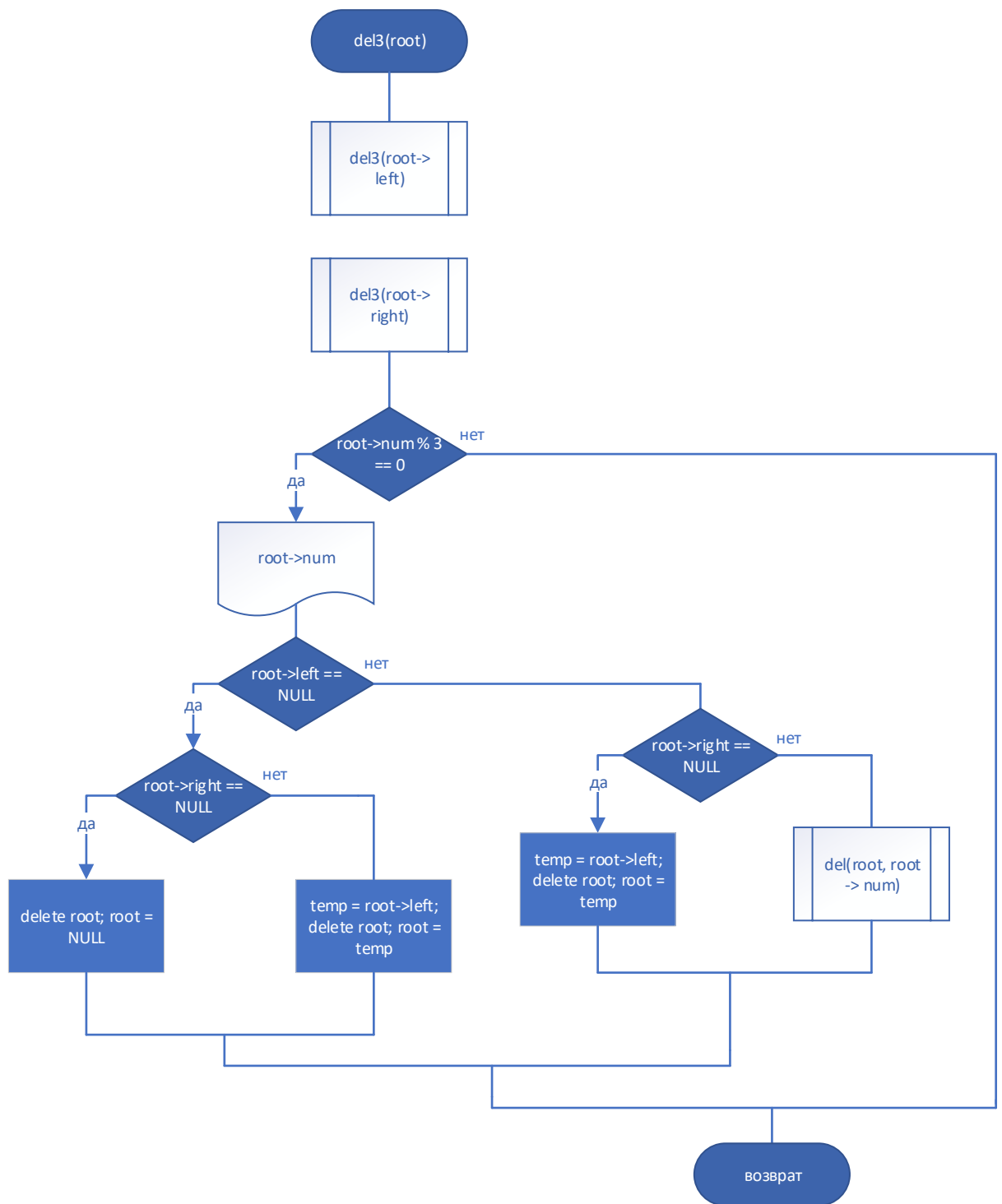


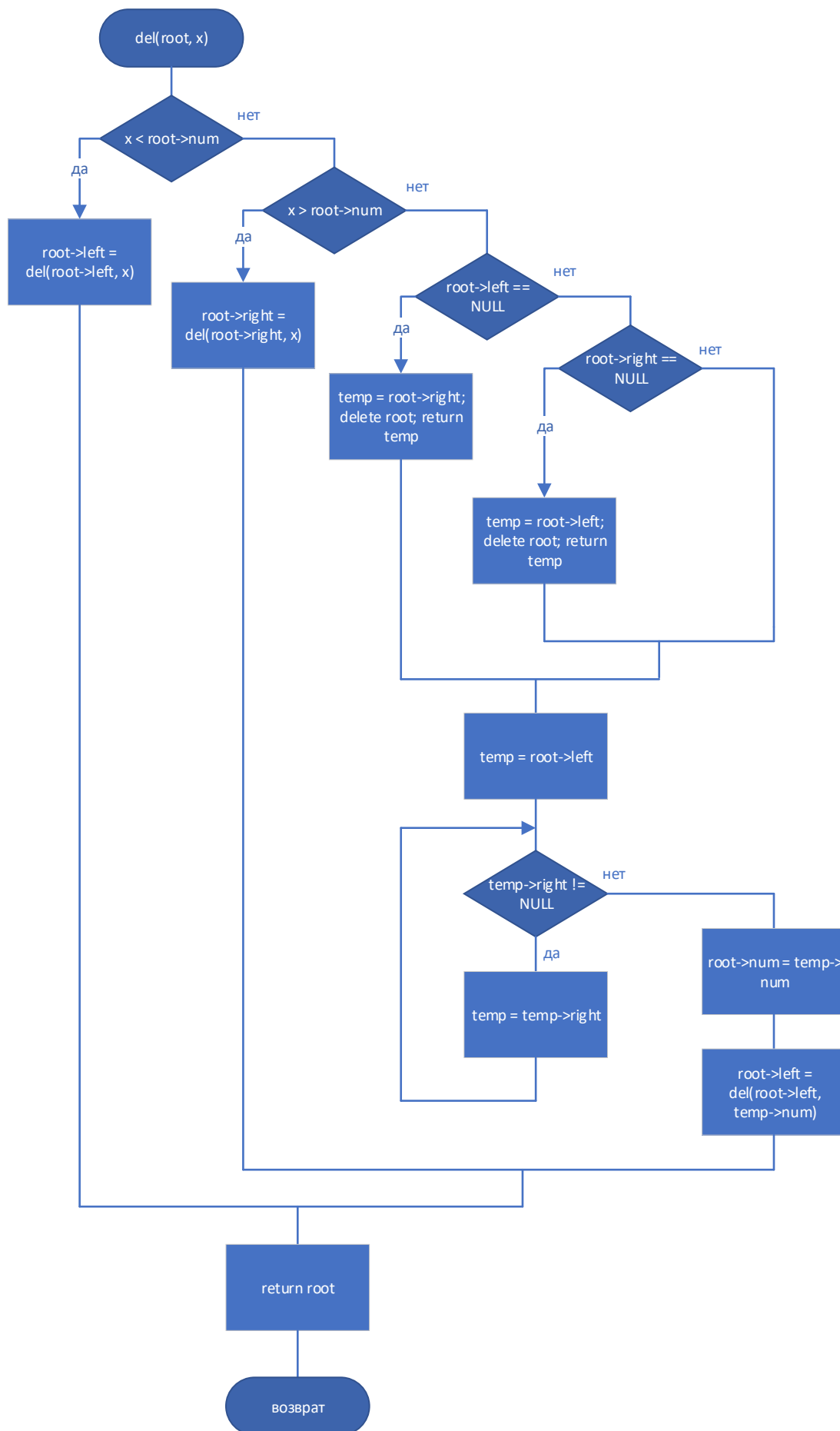
Схемы алгоритмов











Текст программы

```
#include <iostream>

struct Tree {
    int num;
    Tree* left;
    Tree* right;
};

void ins(Tree* cur, Tree* &root) {
    if (cur != NULL) {
        if (root != NULL) {
            if (cur->num < root->num) {
                if (root->left != NULL)
                    ins(cur, root->left);
                else
                    root->left = cur;
            }
            else {
                if (root->right != NULL)
                    ins(cur, root->right);
                else
                    root->right = cur;
            }
        }
        else
            root = cur;
    }
}

void insert(int x, Tree*& root) {
    Tree* cur = new Tree;
    cur->num = x;
    cur->left = NULL;
    cur->right = NULL;
    ins(cur, root);
}

Tree* createTree() {
    Tree* root = NULL;
    std::cout << "Input elements of tree\n";
    char* str = new char[64];
    while (gets_s(str, 64), strlen(str) != 0)
        insert(atoi(str), root);
    delete[] str;
    std::cout << std::endl;
    return root;
}

void strprint(Tree* root) {
    if (root != NULL) {
        std::cout << root->num << " ";
        strprint(root->left);
        strprint(root->right);
    }
}

void symprint(Tree* root) {
    if (root != NULL) {
        symprint(root->left);
    }
}
```



```

        std::cout << root->num << " ";
        symprint(root->right);
    }
}

void revprint(Tree* root) {
    if (root != NULL) {
        revprint(root->left);
        revprint(root->right);
        std::cout << root->num << " ";
    }
}

void print_str(Tree* root) {
    if (root != NULL) {
        std::cout << "Straight printing: ";
        strprint(root);
        std::cout << std::endl;
        std::cout << "Symmetric printing: ";
        symprint(root);
        std::cout << std::endl;
        std::cout << "Reverse printing: ";
        revprint(root);
        std::cout << std::endl;
    }
}

void print_tree(Tree* root, int level) {
    if (root != NULL) {
        print_tree(root->right, level + 4);
        for (int i = 0; i < level; ++i) std::cout << " ";
        std::cout << root->num << std::endl;
        print_tree(root->left, level + 4);
    }
}

void print(Tree* root) {
    print_str(root);
    std::cout << std::endl;
    print_tree(root, 0);
    std::cout << std::endl;
}

Tree* del(Tree*& root, int x) {
    if (root == NULL) {
        return NULL;
    }
    if (x < root->num) {
        root->left = del(root->left, x);
    }
    else if (x > root->num) {
        root->right = del(root->right, x);
    }
    else {
        if (root->left == NULL) {
            Tree* temp = root->right;
            delete root;
            return temp;
        }
        else if (root->right == NULL) {
            Tree* temp = root->left;
            delete root;

```

```

        return temp;
    }
    Tree* temp = root->left;
    while (temp->right != NULL)
        temp = temp->right;
    root->num = temp->num;
    root->left = del(root->left, temp->num);
}
return root;
}

int del3(Tree*& root) {
    static int k = 0;
    if (root != NULL) {

        del3(root->left);
        del3(root->right);

        if (root->num % 3 == 0) {
            std::cout << root->num << " is a multiple of 3\n";
            ++k;
            if (root->left == NULL) {
                if (root->right == NULL) {
                    delete root;
                    root = NULL;
                }

                else {
                    Tree* temp = root->right;
                    delete root;
                    root = temp;
                }
            }

            else {
                if (root->right == NULL) {
                    Tree* temp = root->left;
                    delete root;
                    root = temp;
                }

                else {
                    del(root, root->num);
                }
            }
        }
    }
    return k;
}

void deltree(Tree* &root) {
    if (root != NULL) {
        deltree(root->left);
        deltree(root->right);
        delete root;
        root = NULL;
    }
}

int depth(Tree* root) {
    if (root == NULL) {
        return 0;
    }
}

```

```

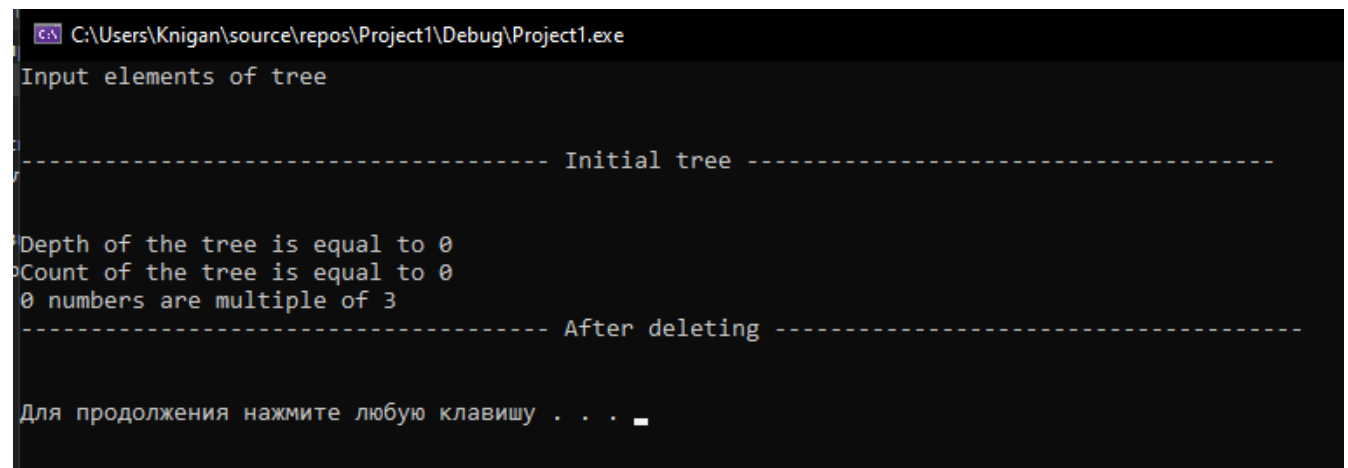
    }
    else {
        int l = depth(root->left);
        int r = depth(root->right);
        if (l > r)
            return l + 1;
        else
            return r + 1;
    }
}

int count(Tree* root) {
    if (root == NULL)
        return 0;
    return count(root->left) + count(root->right) + 1;
}

void main() {
    Tree* root = createTree();
    std::cout << "----- Initial tree -----\\n";
    print(root);
    std::cout << "Depth of the tree is equal to " << depth(root) << std::endl;
    std::cout << "Count of the tree is equal to " << count(root) << std::endl;
    int k = del3(root);
    std::cout << k << " numbers are multiple of 3\\n";
    std::cout << "----- After deleting -----\\n";
    print(root);
    deltree(root);
    system("pause");
}

```

Тестирование



```

C:\Users\Knigan\source\repos\Project1\Debug\Project1.exe
Input elements of tree

----- Initial tree -----

Depth of the tree is equal to 0
Count of the tree is equal to 0
0 numbers are multiple of 3
----- After deleting -----

Для продолжения нажмите любую клавишу . . .

```

C:\Users\Knigan\source\repos\Project1\Debug\Project1.exe

Input elements of tree

21
9
36
5
17
29
8
14
19
24
32

----- Initial tree -----

Straight printing: 21 9 5 8 17 14 19 36 29 24 32

Symmetric printing: 5 8 9 14 17 19 21 24 29 32 36

Reverse printing: 8 5 14 19 17 9 24 32 29 36 21

```

    36
      32
    29
      24
21
    19
    17
      14
    9
      8
    5
```

Depth of the tree is equal to 4

Count of the tree is equal to 11

9 is a multiple of 3

24 is a multiple of 3

36 is a multiple of 3

21 is a multiple of 3

4 numbers are multiple of 3

----- After deleting -----

Straight printing: 19 8 5 17 14 29 32

Symmetric printing: 5 8 14 17 19 29 32

Reverse printing: 5 14 17 8 32 29 19

```

    32
    29
19
    17
      14
    8
    5
```

Для продолжения нажмите любую клавишу . . .

C:\Users\Knigan\source\repos\Project1\Debug\Project1.exe

Input elements of tree

33
48
24
15
36
81
9

----- Initial tree -----

Straight printing: 33 24 15 9 48 36 81

Symmetric printing: 9 15 24 33 36 48 81

Reverse printing: 9 15 24 36 81 48 33

```
      81
     48
    36
33
   24
    15
     9
```

Depth of the tree is equal to 4

Count of the tree is equal to 7

9 is a multiple of 3

15 is a multiple of 3

24 is a multiple of 3

36 is a multiple of 3

81 is a multiple of 3

48 is a multiple of 3

33 is a multiple of 3

7 numbers are multiple of 3

----- After deleting -----

Для продолжения нажмите любую клавишу . . .

```

----- Initial tree -----
Straight printing: 33 18 17 12 3 0 2 9 16 15 14 24 24 32 90 66 34 48 64 81 71
Symmetric printing: 0 2 3 9 12 14 15 16 17 18 24 24 32 33 34 48 64 66 71 81 90
Reverse printing: 2 0 9 3 14 15 16 12 17 32 24 24 18 64 48 34 71 81 66 90 33

```

```

    90
      81
        71
          66
            64
              48
                34
                  32
                    24
                      18
                        17
                          16
                            15
                              14
                                12
                                  9
                                    3
                                      2
                                        0

```

```

Depth of the tree is equal to 7
Count of the tree is equal to 21
0 is a multiple of 3
9 is a multiple of 3
3 is a multiple of 3
15 is a multiple of 3
12 is a multiple of 3
24 is a multiple of 3
24 is a multiple of 3
18 is a multiple of 3
48 is a multiple of 3
81 is a multiple of 3
66 is a multiple of 3
90 is a multiple of 3
33 is a multiple of 3
13 numbers are multiple of 3

```

```

----- After deleting -----
Straight printing: 32 17 2 16 14 64 34 71
Symmetric printing: 2 14 16 17 32 34 64 71
Reverse printing: 14 16 2 17 34 71 64 32

```

```

    71
      64
        34
          32
            17
              16
                14
                  2

```

Для продолжения нажмите любую клавишу . . . █

Тест с добавлением вершины в дерево после его создания

```
C:\Users\Knigan\source\repos\Project1\Debug\Project1.exe
Input elements of tree
33
48
24
15
36
81
9

----- Initial tree -----
Straight printing: 33 24 15 9 48 36 81
Symmetric printing: 9 15 24 33 36 48 81
Reverse printing: 9 15 24 36 81 48 33

      81
     48
    36
33   24
    15
     9

Input the number that will be added to the tree
17
Straight printing: 33 24 15 9 17 48 36 81
Symmetric printing: 9 15 17 24 33 36 48 81
Reverse printing: 9 17 15 24 36 81 48 33

      81
     48
    36
33   24
    15  17
     9

Depth of the tree is equal to 4
Count of the tree is equal to 8
9 is a multiple of 3
15 is a multiple of 3
24 is a multiple of 3
36 is a multiple of 3
81 is a multiple of 3
48 is a multiple of 3
33 is a multiple of 3
7 numbers are multiple of 3
----- After deleting -----
Straight printing: 17
Symmetric printing: 17
Reverse printing: 17

17

Для продолжения нажмите любую клавишу . . .
```

Вывод: я изучил принципы и приёмы работы с бинарными деревьями

Ответы на контрольные вопросы:

1. Что такое дерево?

Дерево – это структура данных, представляющая собой совокупность элементов и отношений, образующих иерархическую структуру этих элементов

2. Что понимают под глубиной дерева?

Высота (глубина) дерева определяется количеством уровней, на которых располагаются его вершины. Высота пустого дерева равна нулю, высота дерева из одного корня – единице. На нулевом уровне дерева может быть только одна вершина – корень дерева, на первом – потомки корня дерева, на втором – потомки потомков корня дерева и т.д.

3. Как выделяется память для представления деревьев?

Списочное представление деревьев основано на понятии элемента, с помощью которого описываются соответствующие вершины дерева. Каждый элемент имеет поле данных и два поля указателей: указатель на начало списка потомков вершины и указатель на следующий элемент в списке потомков текущего уровня. Следует обратить внимание, что при таком способе представления дерева обязательно следует сохранять указатель на вершину, являющуюся корнем дерева!

4. Какие бывают типы деревьев?

Существует большое многообразие древовидных структур данных: бинарные (двоичные) деревья, красно-черные деревья, В-деревья, матричные деревья, смешанные деревья и т.д.

5. Какие стандартные операции можно выполнять над деревьями?

Основными операциями, осуществляемыми с бинарными деревьями, являются:

- создание бинарного дерева;
- печать бинарного дерева;
- обход бинарного дерева;
- вставка элемента в бинарное дерево;
- удаление элемента из бинарного дерева;

- проверка пустоты бинарного дерева;
- удаление бинарного дерева;
- поиск вершины по заданному ключу и т.д.

6. Что такое дерево двоичного поиска?

строится по определенным правилам:

- У каждого узла не более двух потомков.
- Любое значение меньше значения узла становится левым потомком или потомком левого потомка.
- Любое значение больше или равное значению узла становится правым потомком или ребенком правого потомка.

7. Какие виды обходов деревьев вы знаете? Чем они отличаются?

При обходе все вершины дерева должны посещаться в определенном порядке.

Существует несколько способов обхода всех вершин дерева. Выделим три наиболее часто используемых способа обхода дерева (см. таблицу 1):

- прямой обход (префиксный обход)
- симметричный обход (инфиксный обход)
- обратный обход (постфиксный обход)